Balancing Flexibility and Security in Adaptive Process Management Systems

Barbara Weber[1], Manfred Reichert[2], Werner Wild[3], and Stefanie Rinderle[4]

[1] Quality Engineering Research Group, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at
[2] Information Systems Group, University of Twente, The Netherlands
m.u.reichert@cs.utwente.nl
[3] Evolution Consulting, Innsbruck, Austria
werner.wild@evolution.at
[4] Dept. Databases and Information Systems, University of Ulm, Germany
rinderle@informatik.uni--ulm.de

Abstract. Process–aware information systems (PAIS) must provide sufficient flexibility to their users to support a broad spectrum of application scenarios. As a response to this need adaptive process management systems (PMS) have emerged, supporting both ad-hoc deviations from the predefined process schema and the quick adaptation of the PAIS to business process changes. This newly gained runtime flexibility, however, imposes challenging security issues as the PMS becomes more vulnerable to misuse. Process changes must be restricted to authorized users, but without nullifying the advantages of a flexible system by handling authorizations in a too rigid way. This paper discusses requirements relevant in this context and proposes a comprehensive access control (AC) model with special focus on adaptive PMS. On the one hand, our approach allows the compact definition of user dependent access rights restricting process changes to authorized users only. On the other hand, the definition of process type dependent access rights is supported to only allow for those change commands which are applicable within a particular process context. Respective AC mechanisms will be key ingredients in future adaptive PMS.

1 Introduction

In order to support a broad spectrum of applications, process-aware information systems (PAIS) must provide sufficient flexibility at run-time [1,2]. First, PAIS should be quickly adaptable to changes of the real-world processes (e.g., due to business reengineering efforts) [3,4,5]. Second, during the execution of individual process instances users must be able to flexibly deviate from the pre-modeled process schema (e.g., by adding or skipping tasks). Such ad-hoc deviations may become necessary to deal with exceptional situations [1,6].

In response to these needs adaptive process management systems (PMS) have emerged during recent years. Examples include ADEPT [1], CBRFlow [7], METEOR [8] and WASA2 [9]. All these PMS aim at the flexible support of changes at the process type and/or the process instance level. This newly gained flexibility, however, imposes challenging security issues as the PMS becomes more vulnerable to misuse. For example, the uncontrolled addition of long-running activities to an ongoing process instance may delay the execution of the whole process. Appropriate access control (AC) mechanisms are thus even more important for adaptive than for traditional PMS [10,11,12,13,14,15,16] to avoid such misuse scenarios. Process changes must be restricted to authorized users only, but without nullifying the advantages of a flexible system by handling authorizations in a too rigid way.

Although there are several approaches for AC in traditional process management systems (PMS), the special requirements of adaptive PMS have not been addressed in a sufficient way so far. Existing approaches either assume that process schemes are modeled only once and then remain unchanged (i.e., covering access rights for the execution of tasks only) (e.g., [13,10]) or they only support the definition of change rights at a very coarse-granular level [17]. This restricted view, however, is not applicable to adaptive PMS, which require adequate access rights for the different kinds of changes. In particular, access rights must be simple to define, easy to maintain and fast to check.

In addition, it must be possible to enforce the execution of particular activities (e.g., due to legal requirements) and to ensure that only activities which are applicable in a specific context can be inserted into a process instance. For a drug procurement process in a hospital, for instance, the insertion of a patient treatment step makes no sense and should thus not be allowed.

Defining process changes requires user experience and is error prone if not supported by suitable tools. Therefore, adaptive PMS should assist the user while performing changes by displaying only those change commands which are applicable in the current context and for which the user holds the necessary access rights.

In the past we developed detailed concepts for the dynamic modification of process instances, for the evolution of process types and for the memorization and the reuse of process instance changes. This work has been done in the ADEPT and CBRFlow projects [1,7,18,19,20], security issues have not been considered in detail so far. In this paper we introduce an advanced AC model which covers the particular requirements of adaptive PMS. The paper is organized as follows: Section 2 covers adaptive PMS and their characteristics. Section 3 describes the requirements for an AC model, Section 4 extends the core RBAC model to meet the specific requirements for adaptive PMS. Section 5 adds process type dependent access rights to the extended RBAC model. Section 6 describes practical issues and Section 7 discusses related work. The paper concludes with a summary and an outlook in Section 8.

2 Adaptive PMS and Their Characteristics

This section describes background information on adaptive PMS as needed for the further understanding of this paper.

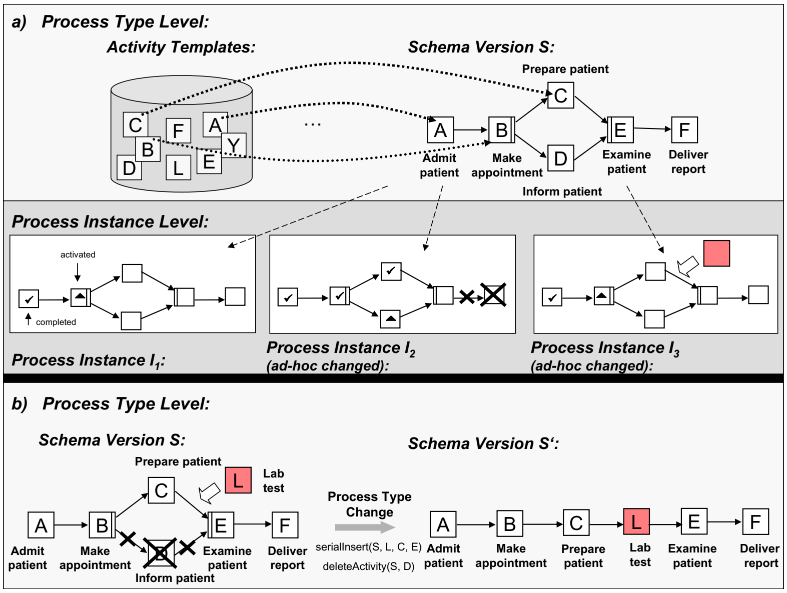


Fig. 1. Different Levels of Process Change (Clinical Example)

2.1 Basic Concepts

In a PMS, for each supported business process (e.g., booking a business trip or handling a medical order) a *process type* T has to be defined. For a particular type one or more *process schemes* (process templates) may exist reflecting different schema versions of T. In Fig. 1 (b), for example, S and S' correspond to different schema versions of the same type. A process schema itself is represented by a directed graph, which consists of a set of *activities* $a_1, \ldots, a_n$ and the *control edges* connecting them. Process schema version S from Fig. 1 (b) consists of six activities; Activity `Admit patient` is followed by activity `Make appointment` in the flow of control, whereas `Prepare Patient` and `Inform Patient` can be processed in parallel. Each activity is based on a predefined activity template, which is maintained and stored in a repository. In general, a particular activity template can be reused within different process schemes.

Based on a schema S new *process instances* $I_1, \ldots, I_m$ can be created and executed at runtime. In Fig. 1 (a), for process instance I_1 activity `Admit patient` has already been completed whereas activity `Make appointment` is currently activated (i.e., it is offered to users in their worklists).

2.2 Process Change

Adaptive PMS are characterized by their ability to correctly and efficiently handle process changes. In general, changes are triggered and performed at two levels – the process type and the process instance level.

Changes to a process type T may become necessary to cover the evolution of real-world business processes [3,4,9]. Process engineers can accomplish a type change by applying a set of change commands to the current schema version S of type T. This results in the creation of a new schema version S' of the same type (cf. Fig. 1 a). Execution of future process instances is then based on S'. For long-running processes it might be necessary to migrate already running process instances to the new schema version S' [18].

By contrast, ad-hoc changes of individual process instances are usually performed by process participants (i.e., end users). Ad-hoc changes are necessary to react to exceptions or unanticipated situations [1,7,8]. The effects of such instance-specific changes are kept local, i.e., they do not affect other process instances of the same type. In Fig. 1 (a) instance I_2 has been individually modified by dynamically deleting activity `Deliver report`. Thus the execution schema of I_2 deviates from the original process schema S of this instance.

In order to facilitate exception handling, adaptive PMS should allow for the memorization and the reuse of ad-hoc deviations. For this, our approach applies case-based reasoning techniques [19,21]. More precisely, changes of individual process instances can be performed either by explicitly defining the change from scratch or by reusing information about previous changes (which were successfully applied to other process instances in similar problem situation before).

In our approach both process type and process instance changes are based on a complete set of change commands with well-defined semantics [1,22]. Table 1 presents selected *high-level change commands* provided in this context.

Table 1. *A Selection of ADEPT Change Commands*[*]

Change Command applied to Schema S	**Effects on Schema S**
	Additive Change Commands
serialInsert(S, X, A, B)	insert activity X into schema S between the two directly connected activities A and B
parallelInsert(S, X, A)	insert activity X into schema S parallel to activity A
	Subtractive Change Commands
deleteActivity(S, X)	delete activity X from schema S
	Order-Changing Commands
serialMove(S, X, A, B)	move activity X from its current position in schema S to the position between two directly connected activities A and B

[*]A detailed description of all change commands supported by ADEPT can be found in [18,22].

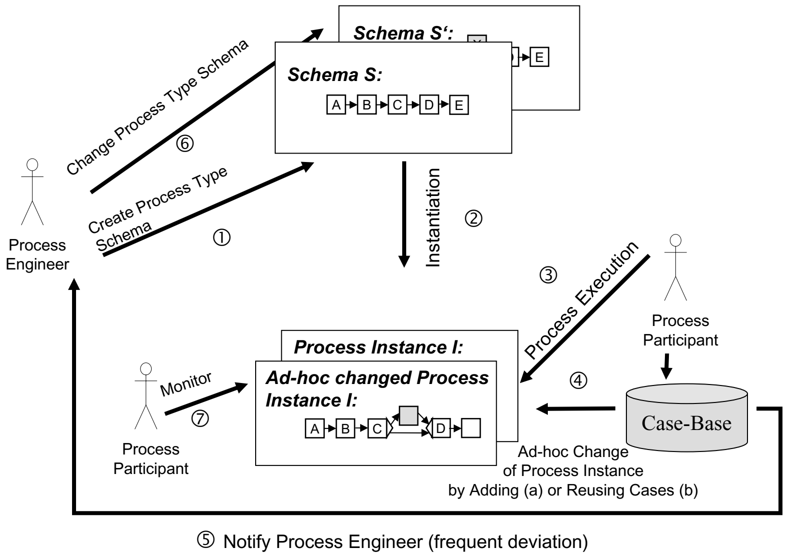

Schema S':
E
Schema S:
A B C D E
Change Process Type Schema
⑥
Create Process Type Schema
①
Process Engineer
Instantiation
②
③
Process Execution
Process Participant
Process Instance I:
Ad-hoc changed Process Instance I:
A B C D
Monitor
⑦
Process Participant
④
Case-Base
Ad-hoc Change of Process Instance by Adding (a) or Reusing Cases (b)
⑤ Notify Process Engineer (frequent deviation)

Fig. 2. Major Use Cases for an Adaptive PMS

2.3 Major Use Cases for an Adaptive PMS

In order to construct a comprehensive AC model for adaptive PMS we must focus on the major use cases of such a system in detail [20]. An overview is given in Fig. 2. At buildtime an initial computerized representation of a company's business processes is created either by business process analysis or by applying process mining techniques (i.e., by observing process and task executions) (1). At run-time new process instances are created from these predefined process schemes (2). In general, process instances are then executed according to the process type schema they were derived from and activities are allocated to authorized process participants to perform the respective tasks (3). However, when deviations from the predefined schema become necessary at the process instance level (e.g., due to exceptions), process participants must be able to deviate from it. They can either specify a new ad-hoc deviation and document the reasons for the changes in a case-base (4 a), or they can reuse a previously specified ad-hoc modification from the case-base (4 b). The PMS monitors how often a particular schema is instantiated and how frequently deviations occur. When a particular ad-hoc modification is frequently reused, the process engineer is notified that a process type change should be performed (5). The process engineer can then evolve the process type schema, and, as far as possible, migrate running instances to the new schema version (6). During run-time process instances can be monitored by process participants (7). Finally, in addition to these use cases, all PMS must support granting access rights.

3 Requirements for an AC Model for Adaptive PMS

To our best knowledge most existing AC models for PMS [10,11,12,13,14,15,16] ignore the problem of process change and therefore do not meet the specific requirements posed by adaptive PMS (cf. Section 7). In this section specific requirements for AC models in adaptive PMS are elaborated. All requirements stem from real world case studies in the medical domain [23].

Requirement 1 (Support of user dependent and process type dependent access rights). An AC model for adaptive PMS should support the definition of both *user dependent* and *process type dependent* access rights in an integrated way. While the former restrict access to authorized users in order to avoid misuse (e.g., only users with role *physician* are authorized to insert the *X-ray* activity), the latter are applied to only allow for change commands that are useful within a particular context (e.g., activity *vacation request* must not be inserted in medical treatment processes).

Requirement 2 (Completeness of the AC model). In order to adequately support adaptive processes it is not sufficient to only provide access rights for executing activities. In addition, all presented use cases (cf. Fig. 2) must be covered. Furthermore, the rights to change process types and process instances must be granted separately, as, for example, a user who is allowed to change a specific process instance usually is not authorized to change the process type schema as well. As another example consider the introduction of a process instance change by reusing information about a previously defined ad-hoc modification: authorization for this use case does not necessarily imply that respective users are also authorized to define a new ad-hoc change from scratch.

Requirement 3 (Fine-grained definition of access rights). In general, it must be possible to specify access rights for individual change commands or groups of change commands (e.g., a particular role is only allowed to insert additional process activities, but not to delete existing ones). In any case, an AC model for adaptive PMS must allow the definition of access rights for all change commands and their parameterizations. For example, a *physician* is authorized to insert additional activities. However, this authorization may be restricted to medical treatment activities (e.g., X-ray, Computer Tomography) and selected processes (e.g., patient examination).

Requirement 4 (Usability and maintainability of access rights). A significant challenge is to balance flexibility and security in such a way that the advantages provided by adaptive PMS are not nullified by a too rigid AC model. Thus, access rights themselves should be simple to define and easy to maintain. In order to support the easy and compact definition of access rights, objects should be hierarchically composed and allow for the definition of access rights at different levels of granularity. For instance, it might be reasonable to authorize a particular role to perform process type changes for all process type schemes supported by the PMS. However, in corporations with a large number of process type schemes different users might be responsible for individual process type schemes or for groups of process type schemes.

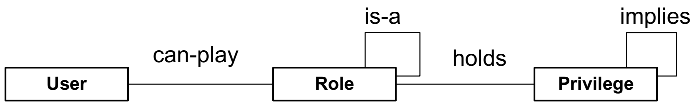


Fig. 3. Core Access Control Model

4 User Dependent Access Rights

An AC model for (adaptive) PMS must allow the system administrator to restrict access to authorized users to avoid misuse. In Section 4.1 we first review basic properties of the core RBAC model, in Section 4.2 and Section 4.3 we then derive an extended role-based AC model to meet the specific requirements of adaptive PMS.

4.1 Core AC Model

The RBAC model is frequently used to specify access rights in PMS [24,25,26]. Access rights are not directly linked to concrete users, but to the more abstract concept of a role. Such roles group privileges (i.e., classes of access rights) and are assigned to users based on their capabilities and competences. *Physician*, *nurse*, and *technician* are examples for roles in a hospital. The role *physician* may include the privileges to order a lab test or to perform a medical examination. Users possessing the same role are considered as being interchangeable, i.e., all users holding a particular role qualify for the privileges associated with that role. A user can hold several roles. In addition, roles can be hierarchically organized, which allows to (transitively) propagate the privileges associated with a particular role to the more specific roles. A head nurse, for instance, is a nurse and therefore inherits all privileges of role *nurse*. Finally, privileges themselves are organized hierarchically to foster inheritance of access rights. The privilege to order restricted drugs (e.g., morphine) up to quantity 1000 implies the more restricting privilege to order such drugs up to quantity 50. Formally, a RBAC model [10] consists of a set of users U, a set of roles R, and a set of privileges P as well as the relationships between elements of these sets (cf. Fig. 3).

In the following the entities and relationships from Fig. 3 are described in more detail.

- A user $u \in U$ represents an individual actor.
- A role $r \in R$ denotes a grouping of privileges which can be assigned to one or more users.
- A privilege $p \in P$ represents a class of rights, e.g., to perform certain operations or to access certain data.
- *can-play* (u, r) states that user u holds role r.
- *is-a* $(r_1, r_2), r_1, r_2 \in R$ states that role r_1 specializes role r_2 and thus inherits all privileges from r_2.

– $holds(r, p), r \in R, p \in P$ states that role r holds privilege p.
– $implies(p_1, p_2), p_1, p_2 \in P$ states that privilege p_1 includes privilege p_2.

4.2 Extended Access Control Model

In this section we extend the core RBAC model by adding additional entities and relationships to construct an adequate AC model for adaptive PMS, which meets Requirements 1-4 of Section 3.

Operation. AC models for adaptive PMS must cover all use cases from Fig. 2 (e.g., changing processes, executing process activities, monitoring process instances, etc.) and be able to grant access rights to each of them separately. We therefore add the entity *operation* to the core RBAC model introduced above.

As illustrated in Fig. 4 (a) we distinguish between seven major operations: *ChangeProcess, CreateSchema, ExecuteActivity, GrantPrivilege, InstantiateSchema, MonitorProcessInstance* and *NotifyUser*. The abstract operation *ChangeProcess* is further divided into process type and process instance changes. Process instance changes can further be split into two groups depending on whether information about a previously defined ad-hoc modification is reused or a new change has to be defined from scratch.

Operations are hierarchically organized, so privileges for a particular operation are automatically extended to their decendents as well (*include* relationship in Fig. 7). As illustrated in Fig. 4 (a), a user holding the privilege for abstract operation *ChangeProcess* is authorized to perform changes at both the process type and the process instance level. The latter can be handled either by adding new or by reusing existing ad-hoc change cases. In contrast, the operation *ReuseExistingProcessInstanceChange* only authorizes for process instance changes based on the reuse of previously defined ad-hoc modifications, but not to define new ad-hoc changes.

Change Command. Although the concept *operation* allows for separate access rights for process type and process instance changes, it does not differentiate between change commands. However, this is indispensable for AC in adaptive PMS as a user might be authorized to skip an activity, but not to perform structural changes by inserting an additional activity. Therefore, we further extend our model with the entity *change command.*

In adaptive PMS both process type and process instance changes can be accomplished by applying a set of well–defined change commands (cf. Fig. 4 b)[1]. When applying the command *serialInsert* an additional activity is inserted between two succeeding process activities. For skipping a particular activity, for example, the command *deleteActivity* can be used. Using different kinds of change commands requires different levels of user experience, which should be taken into account when defining privileges for process change. For example, the deletion of a particular process activity during runtime is always limited in scope and

[1] For illustration we use the ADEPT change commands; however, the model is applicable to other command sets as well.

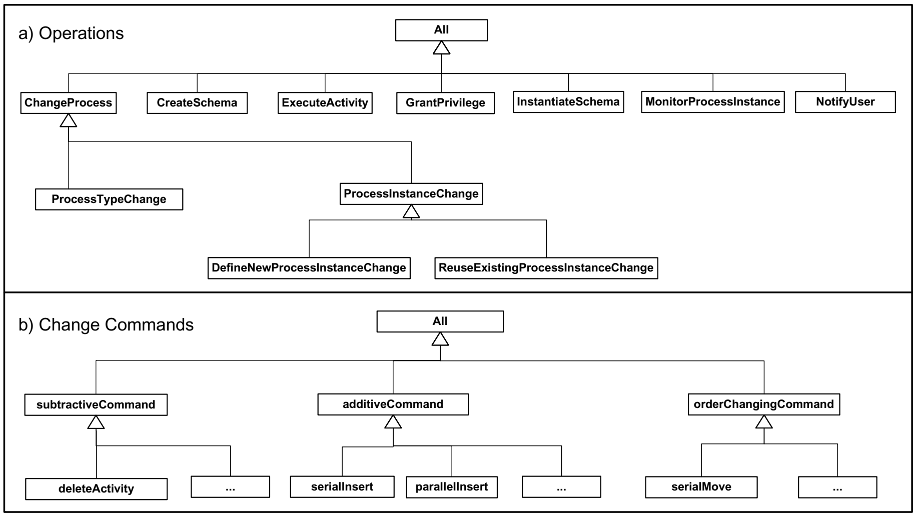


Fig. 4. Operations and Change Commands

can therefore easily be accomplished by end users (i.e., only a few parameters have to be specified when defining the change). In contrast, the insertion of an activity usually requires more comprehensive parameter specifications (e.g., parameters specifying the position of the newly inserted activity) and is therefore more complex to handle.

In order to be able to define access rights at different levels of granularity we allow for the hierarchical organization of change commands (*specializes* relationship in Fig. 7). As illustrated in Fig. 4 (b), the abstract change command *All* includes all kinds of subtractive, additive and order changing commands. If a user holds the right to perform this abstract change command he automatically inherits the right to perform all change commands lower in the hierarchy as well (e.g., *serialInsert, deleteActivity*). On the one hand this approach allows us to define privileges in a very compact way. On the other hand we are able to provide fine-grained specifications as well.

Object. So far we have only considered operations and different change commands. However, we not only must be able to express that a certain role is allowed to delete or add process activities, but also to state which object a particular operation or change command uses (e.g., the process instance that may be monitored or the activity that may be added by an insert command), i.e., the parameterization of operations and change commands has to be considered as well. Therefore we introduce the entity *object* as another dimension in our AC model (cf. Fig 5). For instance, while examining a patient a physician can insert an additional activity `X-ray` (*object).*

Objects are hierarchically organized to achieve maintainable models (*containsObject* relationship in Fig. 7). As illustrated in Fig. 5 access rights can be

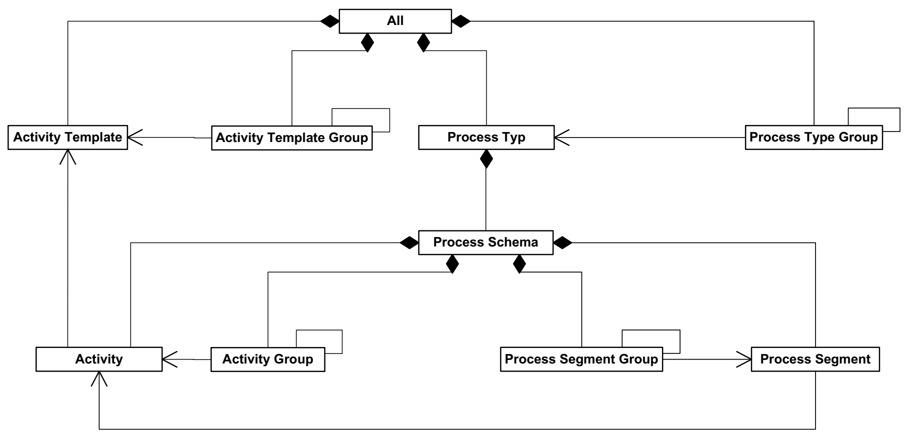


Fig. 5. Object Hierarchy

defined, for instance, for the whole PMS (least detailed level) down to the granularity of single process types or activities (most detailed level). As illustrated in Fig. 6 a particular role holds the privilege to perform the *deleteActivity* command for process type T_1 (=object). Thus this role is authorized to delete all activities which are part of process schemes related to T_1, i.e., activities a_{11}, a_{12}, a_{21} and a_{311}.

Note that a particular operation may not be applicable to all kinds of objects. Table 2 summarizes which combinations of operations and objects are applicable. For example, the *ChangeProcess* operation in combination with an order-changing change command can be used with the following objects: process

Table 2. Granularity of Objects Depending on the Operation*

Operation	All	T	PTG	S	PS	PSG	A	AG	AT	ATG
Change Process										
- additive change									X	X
- subtractive change	X	X	X	X	X	X	X	X	X	X
- order-changing change	X	X	X	X	X	X	X	X		
Create Schema	X		X							
Execute Activity							X	X		
Grant Privilege	X	X	X	X	X	X	X	X	X	X
Instantiate Schema	X	X	X	X						
Monitor Process Instance	X	X	X	X	X	X				
Notify User	X	X	X	X	X	X				

*All = Process Management System, T = Process Type Schema, PTG = Process Type Group, S = Process Schema Version, PS = Process Segment, PSG = Process Segment Group, A = Activity, AG = Activity Group, AT = Activity Template, AT = Activity Template Group.

management system (*All*), process type, process type group, process schema version, process segment, process segment group, activity and activity group. In contrast, for the *ChangeProcess* operation in combination with additive change commands only activity templates or groups of activity templates can be used as the object.

Subject. Finally, we introduce the entity *subject* to specify what is subject to change. For example, when an additional lab test activity (*object*) is inserted (*change command*), we must know the process schema version or the process segment it will be added to, i.e., the subject for this insertion command must be specified.

Like objects, subjects are organized in a hierarchy too (*containsSubject* relationship in Fig. 7). When specifying subjects only a sub-set of the elements from Fig. 5 is required. Candidates for subjects are the whole process management system, process types, groups of process types, process schema versions and process segments; however, activities, groups of activities, activity templates and groups of activity templates are not applicable.

As illustrated in Fig. 6, if a particular role has the privilege to perform process instance changes (=operation) using the *serialInsert* command for activity template at_1 (=object) and subject T_1, then this role is authorized to insert activity template at_1 into process instances based on process type schema T_1.

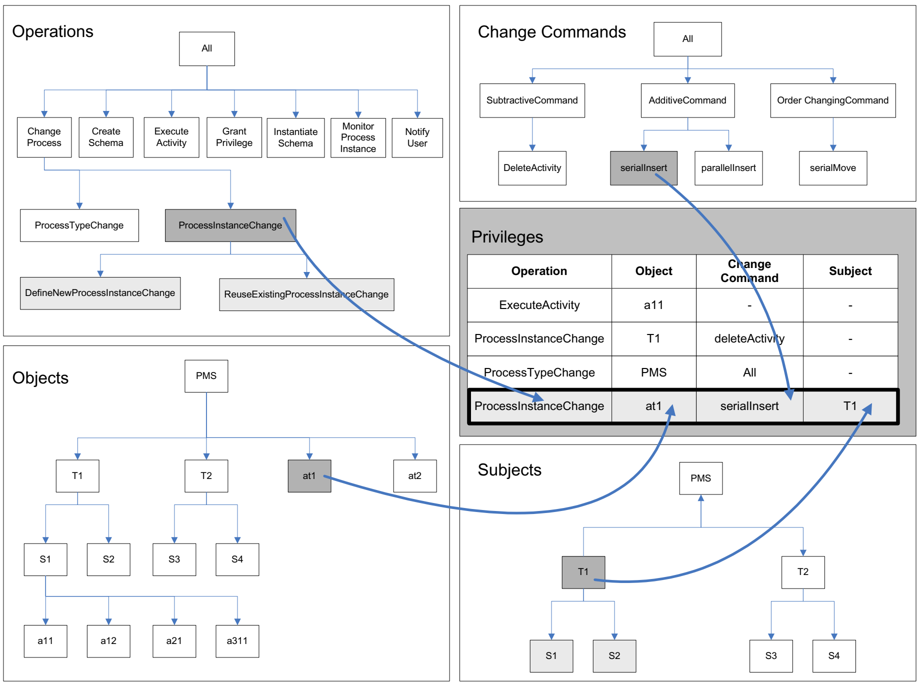
Specifying a subject is only needed for *additive* change commands. For all other operations no subject must be specified as the subject is automatically known by the system, due to the containment relationships (cf. Fig. 5). When, as illustrated in Fig. 6, the object for operation *ExecuteActivity* is activity a_{11}, then the subject is implicitly known to be S_1, the process schema version activity a_{11} is part of.

Privilege. As illustrated in Fig. 6 a privilege is defined by specifying an operation, an object, a change command, and a subject. For example, privilege *(ProcessInstanceChange, at_1, serialInsert, T_1)* states that activities derived from activity template at_1 can be added to any process instance created from process type T_1. Fig. 6 shows the field values of the selected privilege in dark grey. The selected privilege automatically extends to the light grey boxes as well; for example, in Fig. 6 the subject is T_1, this extends the privilege to process schema versions S_1 and S_2 too.

As indicated in Fig. 6 not all entities are mandatory in all situations. The entities *operation* and *object* are always mandatory, while the entity *change command* is only mandatory when the *ChangeProcess* operation is selected. The entity *subject* is only mandatory for *additive* change commands.

4.3 Extended Model - Overview

We added the additional entities *operation* OP, *object* O, *change command* C and *subject* SUB as well as the relationships *includes*, *containsObject*, *specializes* and *containsSubject* to the core RBAC model. In addition the entity *privilege*



Operation	Object	Change Command	Subject
ExecuteActivity	a11	-	-
ProcessInstanceChange	T1	deleteActivity	-
ProcessTypeChange	PMS	All	-
ProcessInstanceChange	at1	serialInsert	T1

Fig. 6. Extended Access Control Model - Example

P has been defined in a more detailed way. The meta-model of the extended AC model is illustrated in Fig. 7.

- Object $o \in O$ represents one of the following entities: the entire process management system, a process type, groups of process types, a process schema, a process segment, groups of process segments, an activity, a group of activities, an activity template or a group of activity templates.
- Operation $op \in OP$ represents a use case supported by the adaptive PMS (e.g., *ExecuteActivity*, *GrantPrivilege* or *ChangeProcess*).
- Change Command $c \in C$ represents a change command (e.g., *deleteActivity or serialInsert*).
- Subject $sub \in SUB$ represents what is subject to change, i.e., the entire process management system, a process type, a group of process types, a process schema, a process segment and a group of process segments.
- Privilege $p \in P$ is a $tuple(op, o, c, sub)$ representing the right to perform a particular operation op with an object o using change command c on a subject sub.
- $includes(op_1, op_2), op_1, op_2 \in OP$ states that operation op_1 includes op_2. Having a privilege for op_1 thus includes the privileges for op_2.
- $containsObject(o_1, o_2), o_1, o_2 \in O$ states that object o_1 includes object o_2.

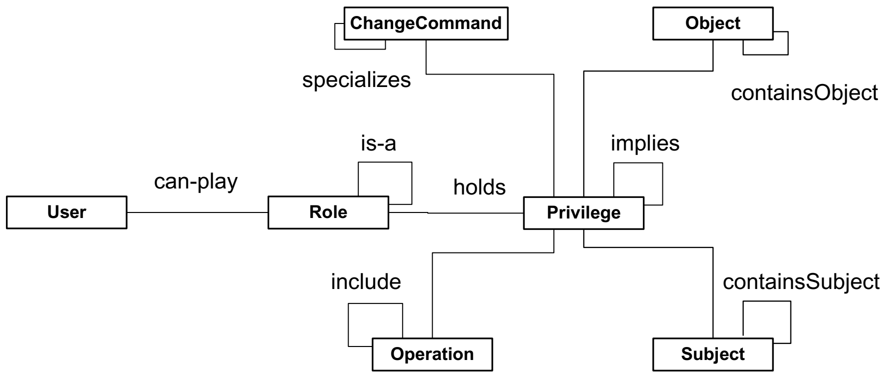


Fig. 7. Extended Access Control Model

– *specializes*$(c_1, c_2), c_1, c_2 \in C$ states that change command c_1 includes change command c_2.
– *containsSubject*$(sub_1, sub_2), sub_1, sub_2 \in Sub$ states that subject sub_1 includes subject sub_2.

5 Process Type Dependent Constraints

An AC model for adaptive PMS must not only allow to restrict access to authorized users. It must also enforce the execution of particular activities (e.g., a particular activity must not be deleted due to legal requirements) and ensure that only semantically correct activities can be inserted in the given context (e.g., no patient related activities must be inserted into a drug procurement process).

Process type dependent access rights allow to specify which activities, activity templates, which groups of activities and which groups of activity templates can be inserted into, deleted from or moved within process instances based on a particular process type.

Process type dependent access rights are defined independently of the individual users and roles performing the process instance change, thus only a sub-set of the extended AC model in Fig. 7 is needed; the entities *role* and *user* can be omitted. Like user dependent access rights, a privilege for process type dependent access rights consists of the entities *operation*, *object*, *change command* and *subject*.

Example 1. Privilege *(ProcessInstanceChange, MedicalTreatmentStep, additiveCommand, GroupMedicalTreatmentProcess)* says that any activity template of group *MedicalTreatmentStep* (i.e., `X-ray, Lab Test, Computer Tomography`) can be inserted into any medical treatment process by using an additive change command.

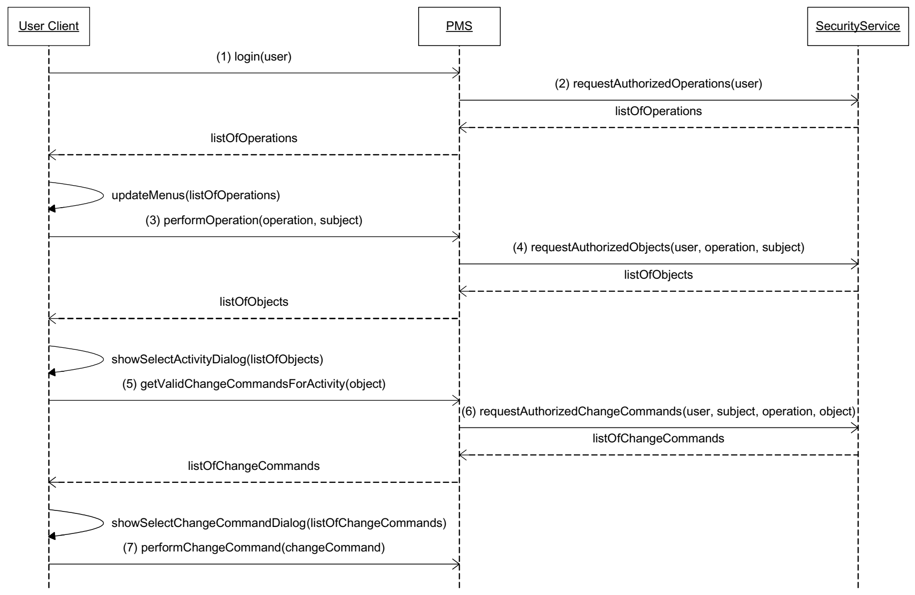

User Client
PMS
SecurityService
(1) login(user)
(2) requestAuthorizedOperations(user)
listOfOperations
listOfOperations
updateMenus(listOfOperations)
(3) performOperation(operation, subject)
(4) requestAuthorizedObjects(user, operation, subject)
listOfObjects
listOfObjects
showSelectActivityDialog(listOfObjects)
(5) getValidChangeCommandsForActivity(object)
(6) requestAuthorizedChangeCommands(user, subject, operation, object)
listOfChangeCommands
listOfChangeCommands
showSelectChangeCommandDialog(listOfChangeCommands)
(7) performChangeCommand(changeCommand)

Fig. 8. PMS and Security Service Interactions (Process Instance Change)

6 Practical Issues

Our extended AC model is currently implemented as a separate Security Service (SECS) which can be used independently of a specific PMS. To demonstrate the interactions between a PMS and the SECS this section walks the reader through a process instance change (cf. Fig. 8).

Dynamic Menu Configuration. When the user logs in to the PMS via his user client (1) the PMS interacts with the SECS to request the list of operations the user is authorized for (2) (Query: *requestAuthorizedOperations(User)*). Based on the results of this query the menu of the user client is dynamically configured, i.e., only those operations are displayed in the user's menu for which he is authorized. Thus, the user is never presented a menu item he is not authorized for, which prevents annoying "not authorized" warnings.

Example 2. The physician *John* logs in to the PMS via his user client. The PMS then dynamically configures the menu items based on the privileges which have been assigned to the role *physician* and which comply with the process type dependent access rights (cf. Fig. 9 a). John, for instance, is authorized to insert activity templates of group *MedicalTreatmentSteps* (i.e., `X-ray, Lab Test, Computer Tomography`) into process instances of *medial treatment processes*.

Perform Ad-hoc Changes. When an authorized user wants to deviate from the predefined process schema at runtime he can select the respective menu item to perform a process instance change in his user client (3).

(a)

User dependent access rights

Operation	Change Command	Object	Subject	Role
ProcessInstanceChange	serialInsert	MedicalTreatmtentSteps	GroupMedicalTreatmentProcess	Physician

Process type dependent access rights

Operation	Change Command	Object	Subject
ChangeProcess	serialInsert	MedicalTreatmtentSteps	GroupMedicalTreatmentProcess

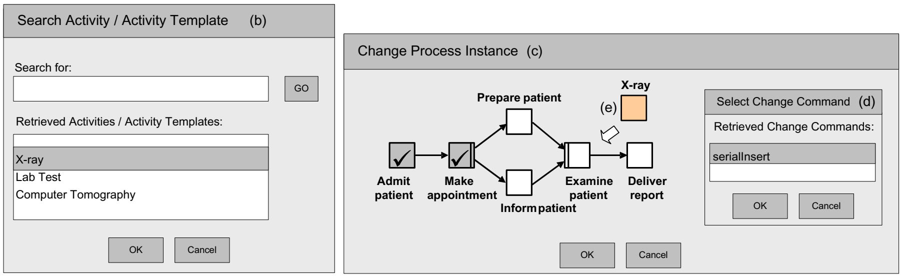


Fig. 9. Example Privileges (a) and User Interactions (b-e)

He then has to select the object for the change, e.g., the activity template to be inserted into the process instance schema or the concrete activity to be deleted or moved (cf. Fig. 9 b).

For insert operations the PMS requests the set of activity templates the user is authorized to insert into process instances based on process schema version S (4). It further requests the set of activities in process schema S the user is authorized to delete or move and which comply with the process type dependent access rights (Query: *requestAuthorizedObjects(User, Subject, Operation)*). Completed activities are not listed, as already passed process graph regions can no longer be modified. The user can then select one of the displayed activities or activity templates (5).

Example 3. Assume that John decides to perform a new ad-hoc modification (e.g., to insert a new activity based on template `X-ray`) for a process instance created from process schema version $S1$ (cf. Fig. 9 b). He selects the respective operation in the menu bar of his user client, and the system then requests the list of activity templates (*requestAuthorizedObjects(John, S1, ProcessInstanceChange)*) he is authorized to add to this schema. In our example, the SECS returns activity templates `X-ray`, `Lab Test`, and `Computer Tomography`.

The system then shows a graphical representation of process schema S (cf. Fig. 9 c) and suggests those change commands to the user which comply with both his authorizations and the process type dependent access rights (6) (Query: *requestChangeCommands(User, Operation, Object, Subject)*). The user can then choose one of the displayed change commands for execution (7). When necessary, the system requests any required parameters from the user (cf. Fig. 9 d).

Example 4. John selects activity `X-ray`. The PMS then displays $S1$, the process schema on which the process instance to be modified was created from.

After this, the PMS requests a list of change commands which are presented to John (*requestChangeCommands(John, ProcessInstanceChange, X-ray, S1)*). John then selects the *serialInsert* change command to insert the additional `X-ray` activity. Furthermore, the system asks him where the `X-ray` activity should be inserted and ensures that he does not insert it into already passed process graph regions. John replies that the `X-ray` activity is to be performed after activity `Examine Patient` and before activity `Deliver Report` (cf. Fig. 9 e).

7 Related Work

There are several AC models for PMS discussed in literature [10,11,12,13,14,15, 16,17]. Most of them use RBAC models and support static as well as dynamic constraints (e.g., [10,11,12,13,14]). However, they only provide limited support for adaptive processes: either they only cover privileges for the execution of tasks, but do not deal with privileges for process changes at all (e.g., [10,13]), or specify change rights at a very coarse-grained level (e.g., not distinguishing between change commands) [17].

W-RBAC [10] provides a specific AC model for workflow systems which focuses on the definition of task execution rights and allows for the definition of static and dynamic constraints (e.g., separation of duties). Dynamic constraints have not been the focus of this paper, but will be considered in our overall approach as well, we plan to apply concepts provided by existing approaches [10,11,12,13,14]. In order to deal with exceptional situations W-RBAC allows authorized users to override constraints. However, as the definition of change rights is not supported, W-RBAC is not suited for adaptive PMS.

Case-handling systems provide an alternative to adaptive PMS. FLOWER [15], for example, allows defining change rights to some degree. For each process and for each activity an *execution role* (for carrying out the activity or to start the process), a *redo role* (to undo activities) and a *skip role* (to omit an activity) can be defined. The handling of more complex change commands (e.g., the insertion of new activities) is not addressed.

Domingos et al. [17] propose an AC model for adaptive workflows, which also considers the evolution of access rights. Though their approach differentiates between process type and process instance change, it does not allow for fine-grained definition of privileges at the level of individual change commands. Their focus is on user dependent access rights, process dependent rights are not considered. As no abstraction mechanisms (e.g., hierarchies) are used, the compact definition of access rights is not possible.

An approach to control flexibility other than by AC is the Pockets of Flexibility model [27]. Flexibility is limited to certain predefined points in the workflow, where the workflow can be dynamically extended at run-time.

8 Summary and Outlook

In this paper we presented an AC model which allows for the compact definition of access rights as needed in adaptive PMS. We support both the definition of user dependent and process type dependent access rights (cf. Requirement 1). Our approach supports the use cases provided by an adaptive PMS (cf. Requirement 2), and allows the specification of access rights for individual change commands (cf. Requirement 3). If desired, access rights can be specified at an abstract (i.e., coarse-grained) level by using the hierarchical organization of our model. Fine-grained specification of access rights is supported as well, allowing context-based assistance of users when performing a change. However, the more detailed the respective specifications, the more costly their definition and maintenance becomes. Based on our experience with processes from several application domains, different granularities must be supported (cf. Requirement 4).

Currently we are working on the implementation of the Security Service and its integration into the adaptive PMS ADEPT [1] and CBRFlow [7]. We further plan a thorough evaluation of the AC model in real world settings, including its performance and scalability. Next, dynamic constraints will be elaborated in more detail and integrated into our AC model.

References

1. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
2. Jørgensen, H.D.: Interactive Process Models. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
3. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering, Special Issue on Advances in Business Process Management **50** (2004) 9–34
4. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering **24** (1998) 211–238
5. v.d. Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoret. Comp. Science **270** (2002) 125–203
6. Strong, D., Miller, S.: Exceptions and exception handling in computerized information processes. ACM–TOIS **13** (1995) 206–233
7. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In: Proc. Eurpean Conf. on Case–based Reasoning (ECCBR'04), Madrid (2004) 434–448
8. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. Applied Intelligence **13** (2000) 125–147
9. Weske, M.: Workflow management systems: Formal foundation, conceptual design, implementation aspects. University of Münster, Germany (2000) Habil Thesis.
10. Wainer, J., Barthelmess, P., Kumar, A.: W-RBAC - a workflow security model incorporating controlled overriding of constraints. IJCIS **12** (2003) 455–485
11. Bertino, E., Ferrari, E., Alturi, V.: The specification and enforcement of authorization constraints in wfms. ACM Trans. on Inf. and Sys. Sec. **2** (1999) 65–104
12. Botha, R., Eloff, J.: A framework for access control in workflow systems. Information Management and Computer Security. **9** (2001) 126–133

13. Casati, F., Castano, S., Fugini, M.: Managing workflow authorization constraints through active database technology. Inf. Sys. Frontiers. **3** (2001) 319–338
14. Liu, D.R., Wu, M.Y., Lee, S.T.: Role-based authorization for workflow systems in support of task-based separation of duty. The Journal of Systems and Software. **73** (2004) 375–387
15. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. Data and Knowledge Engineering. **53** (2005) 129–162
16. Atluri, V., Huang, W.K.: Enforcing mandatory and discretionary security in workflow management systems. Journal of Computer Security. **5** (1997) 303–339
17. Domingos, D., Rito-Silva, A., Veiga, P.: Authorization and access control in adaptive workflows. In: ESORICS 2003. (2003) 23–38
18. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, University of Ulm (2004)
19. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution - a semantics based approach. In: BPM 2005. (2005)
20. Weber, B., Reichert, M., Rinderle, S., Wild, W.: Towards a framework for the agile mining of business processes. In: Proc. of Int'l BPI workshop. (2005)
21. Weber, B., Rinderle, S., Wild, W., Reichert, M.: CCBR–driven business process evolution. In: ICCBR'05, Chicago (2005)
22. Reichert, M.: Dynamic Changes in Workflow-Management-Systems. PhD thesis, University of Ulm, Computer Science Faculty (2000) (in German).
23. Konyen, I.: Organizational structures and business processes in hospitals. Master's thesis, University of Ulm, Computer Science Faculty (1996) (in German).
24. Ferraiolo, D., Kuhn, D.: Role based access control. In: 15th National Computer Security Conference. (1992)
25. Sandhu, R.S., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. IEEE Computer **29** (1996) 38–47
26. Ferraiolo, D.F., Chandramouli, R., Kuhn, D.R.: Role-Based Access Control. Artech House, Incorporated (2003)
27. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specifications. In: Proc. Int'l Entity–Relationship Conf. (ER'01), Yokohama (2001) 513–526